An Evaluation of Different Page Allocation Strategies on High-Speed SSDs

Myoungsoo Jung and Mahmut Kandemir
Department of CSE, The Pennsylvania State University
{mj,kandemir@cse.psu.edu}

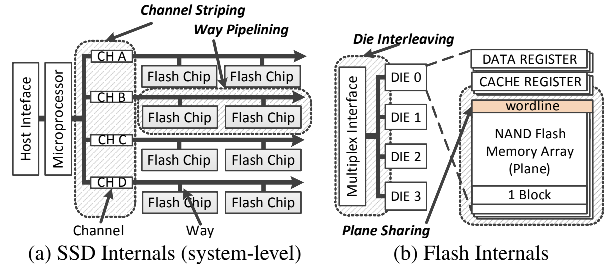
Abstract

Exploiting internal parallelism over hundreds NAND flash memory is becoming a key design issue in high-speed Solid State Disks (SSDs). In this work, we simulated a cycle-accurate SSD platform with twenty four page allocation strategies, geared toward exploiting both system-level parallelism and flash-level parallelism with a variety of design parameters. Our extensive experimental analysis reveals that 1) the previously-proposed channel-and-way striping based page allocation scheme is not the best from a performance perspective, 2) As opposed to the current perception that system and flash-level concurrency mechanisms are largely orthogonal, flash-level parallelism are interfered by the system-level concurrency mechanism employed, and 3) With most of the current parallel data access methods, internal resources are significantly underutilized. Finally, we present several optimization points to achieve maximum internal parallelism.

1 Introduction

NAND flash-based Solid State Disks (SSDs) are being increasingly used in enterprise, personal and high performance computing systems, due to their performance advantage over spinning devices. While high-performance interfaces with transfer rates ranging from 6Gb/sec to 16GT/sec are being adopted by modern SSD architectures, the speed of NAND flash memory (i.e., flash) is still limited by about 40MB/sec [5]. This performance gap between SSD interfaces and flash chips has driven the research that target internal parallelism in SSDs, which can have a great impact on improving system performance. From an architecture perspective, SSD systems and flash devices expose parallelism at different levels. More specifically, SSD systems employ multiple flash chips over multiple channel I/O buses and multiplexed flash interfaces, which means that multiple SSD components can be simultaneously activated to serve incoming I/O requests. In parallel, flash technologies are being developed to extract maximum parallelism. A single flash chip consists of multiple dies, each of which accommodating multiple planes. Obviously, performance characteristics of modern SSDs are varied based on what strategies are employed for parallelizing data accesses across hundreds or thousands of flash dies and planes. A key design issue behind exploiting parallel data accesses is how to efficiently exploit internal parallelism and how to organize parallelism-friendly physical data layout for both the SSD system and flash levels.

As exploiting internal parallelism is key to improving performance and filling the performance gap between flash and high-speed interfaces, parallel data access methods are getting attention from both academia and industry [3, 4, 12, 13]. Architectural approaches to system-level parallelism using multiple I/O buses and flash chips such as ganging/superblock have been explored, and flash-level concurrency mechanisms utilizing



(a) SSD Internals (system-level) (b) Flash Internals

Figure 1: Internals of SSD and NAND flash and illustration of different level data accessing methods.

multiple dies and planes within a flash chip like interleaving/banking have been already studied [2, 5]. However, *page allocation (palloc) strategies* enabling both system- and flash-level parallelism by determining physical data layout, have received little attention so far. A few palloc schemes in favor of the channel striping based data access method have been investigated [8], and the interplay between flash-level parallelism and these *channel-first* palloc schemes have been studied [6]. In addition, very little has been published on understanding the interactions between system-level and flash-level parallelism.

In this work, we explore different page allocation strategies, geared toward exploiting both system-level and flash-level parallelism – we study a full design space siting on system and flash-level organizations with a variety of parameters such as a standard queue, multiple buses, chips, and diverse advance flash operations. Specifically, we evaluate twenty four palloc strategies including the *flash-level resource-first* and *way-first* strategies we defend. The questions we are interested in answering include 1) *which palloc scheme would be globally optimal for parallelizing data accesses when both system- and flash-level parallelisms are considered?*, 2) *what are the relationship between different level concurrency methods?*, and 3) *what are the resource utilizations of different palloc schemes?* To the best of our knowledge, this is the first paper that explores all possible combinations of palloc strategies considering all levels of parallelism in SSDs. Our main **contributions** can be summarized as follows:

• *Determining good page allocation schemes.* We observe from our experiments that the channel-and-way striping based palloc is *not* the best strategy from a performance perspective, despite recent works [6, 8] claiming that. Our experiments in contrast reveal that, when advance flash operations are considered, a flash-level resource-first palloc scheme results in better throughput than the approach in [6, 8] (as much as 84.8% and on average 40.1% with very similar response times).

• *Addressing parallelism interference.* As opposed to the common perception that system and flash-level concurrency mechanisms are largely orthogonal, channel striping method in system-level makes it hard to exploit flash-level parallelism under disk-friendly workloads. In fact,

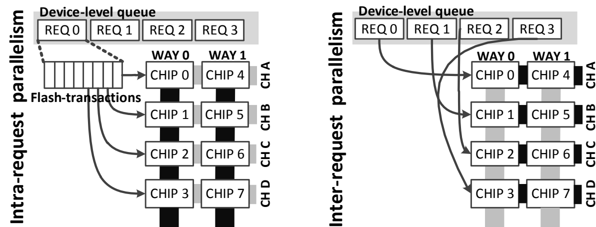


Figure 2: Request-level parallelism.

in the worst case, flash-level parallelism extracted by the channel striping method shrinks as much as 99% and on average 44% since it exhibits poor locality in flash chips.

• *Addressing resource utilization.* We observe from our experimental analysis that most parallel data access methods and palloc schemes have room for performance improvement since many internal resources with them are still underutilized. When considering all the cases tested, channel resources are 57.9% underutilized and the activate time for buses and flash memory cells accounts for only 22.1% of the total execution time.

2 SSD Internals and Parallelisms

Since multiple flash chips are packaged in the form of an SSD, there are numerous hardware components and buses that work in tandem to provide access to the internals of flash. In this context, *channels* are I/O buses that are independently operated by microprocessors, and *ways* are data paths, connected to flash chips in each channel. Within each flash chip are one or more *dies*, sharing the single multiplexed interface. Lastly, the dies accommodate multiple *planes*, the smallest unit to serve a request in parallel. Figures 1(a) and 1(b) depict SSD and flash internals with corresponding *parallel data access methods*, respectively.

System-Level Parallelism. At a system-level, in the beginning of a data access process, an I/O request can be striped over multiple channels, and this process is termed as *channel striping*. Unlike channels, way-level activities should be serialized because the multiplexed interfaces of each flash chip are shared within a channel. Individual chips can however work in parallel, and a flash memory transaction consists of multiple phases; consequently, I/O requests can be pipelined. Therefore, using *way pipelining*, multiple I/O requests can be simultaneously served by multiple flash chips in a channel.

Flash-Level Parallelism. After I/O requests are striped over multiple flash chips, they can be further interleaved across multiple dies in a flash chip. Similar to way pipelining, the data movements and flash command controls in this *die interleaving* need to be serialized. Still, in an ideal case, performance increases by about n times, where n is the number of dies. *Plane sharing* concurrently activates flash operations on multiple planes, which can improve performance by about m times, where m is the number of planes. Finally, these two parallel data access methods can be combined when incoming I/O requests span all of the flash internal components. This method is referred to as *die interleaving with multiplane*, and it can improve performance by about $n{*}m$ times. It should be noted however that, unlike system-level parallelism, data accessing mechanisms in this level are only available via *advance flash commands* provided by flash chip makers.

Request-Level Parallelism. Parallel data access methods can serve flash-transactions within an I/O request or between the I/O requests sitting in a device-level queue (Figure 2). Generally speaking, *Intra-request parallelism*, referring the former reduces latency, and *inter-request parallelism* indicating the latter, improves storage throughput.

3 Page Allocation Strategies

SSDs decide physical data layout by remapping logical and physical addresses. This data layout within and between flash chips should be carefully determined so that one can exploit all levels of parallelism mentioned in Section 2. Since page allocation (*palloc*) strategies are directly related to the physical layout of data, the performance of an SSD can vary based on which palloc scheme is employed.

Figure 3 illustrates twenty four different palloc strategies oriented toward exploiting system-level and flash-level resources. At the top-left corner of Figure 3(a), we show how to identify the internal resources in these different palloc strategies. In order to distinguish among different palloc strategies, we use abbreviations composed of the initial letters of internal resources based on their priority. The order of numbers in the figure indicates how each palloc scheme allocates internal resources. For example, in the CWDP (Channel-Way-Die-Plane) palloc scheme, requests are first striped across multiple channels and ways. Flash-transactions corresponding to these requests are then assigned to multiple dies and planes.

Channel-first palloc strategies allocate internal resources in favor of the channel striping method, which can maximize the benefits coming from intra-request parallelism. Therefore, latencies experienced by these palloc strategies are expected to be lower when the requests span all of channels. In comparison, *way-first palloc strategies* are oriented toward taking advantage of the way pipelining, and can improve throughput by maximizing inter-request parallelism. In contrast, *die-first* and *plane-first palloc strategies* allocate flash-level resources rather than channels or ways in an attempt to reap up the benefits of die interleaving, plane sharing, or die interleaving with multiplane methods.

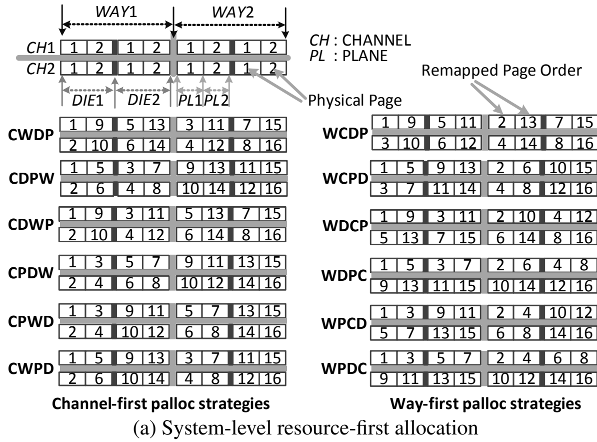
These palloc strategies can be incorporated into an existing flash translation layer (FTL), which is the internal software to perform mapping between logical and physical addresses. Even under the situation that the FTL remaps addresses, page ordering performed by pallocs will still have a great performance impact because pallocs determine the order in which coalesced data pages are written to physical pages, which in turn influences the order in which the pages are read.

4 Experimental Methodology

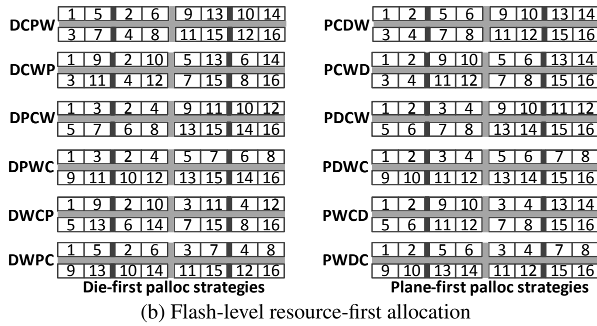
To evaluate each of the palloc schemes shown in Figure 3, we needed a high-fidelity simulator that can capture cycle-level accuracy and interaction between internal resources. Motivated by this, we developed a *cycle-accurate* NAND flash simulator[1], which is hardware-validated, aware of intrinsic flash latency variation and support advance flash operations. Micron multi-level cell (MLC) NAND flash[2] is used for the NAND flash simulator. The package type of this MLC flash is dual die, and it employs a two-plane architecture. We built a simulation framework that com-

[1] The source code of this simulator [9] can be downloaded from http://www.cse.psu.edu/∼mqj5086/nfs.

[2] 2 KB page size, page read latency is 50 μsec, page write latencies are varied from 250 μsec to 2.2 $m$sec, and erase time is 2.5 $m$sec [10].



(a) System-level resource-first allocation



(b) Flash-level resource-first allocation

Figure 3: Different page allocation strategies.

	Data Size (MB)	**Write Fraction (%)**	**Avg. Write Size (KB)**	**Avg. Read Size(KB)**	**Random-ness (%)**
fin1	18057	84.6	1.5	1	96.9
fin2	8846	21.5	1	1	97.4
msnfs	32490	93.9	10	23.5	87.2
usr	50727	27.0	5	20	92.2
web	15985	0.1	4	7.5	93.5
sql0	30433	40.2	4	14.5	89.9
sql1	4676	14.55	4.5	22.5	73.6
sql2	276407	0.3	10.5	26.5	71.9
sql3	1196	37.12	10	37	56.8

Table 1: Important characteristics of our traces.

bines multiple NAND flash simulator instances under a page level address mapping flash translation layer and a garbage collector similar to the one employed in [2]. Eight channels and eight ways are simulated with a FIFO-style NCQ (32 entries on virtual addresses that the FTL provides) [7]. Each channel works at 50 MHz and the frequency of microprocessor used to parallelize data accesses is 800 MHz. For evaluating the effectiveness of our palloc strategies, we chose real enterprise-scale workloads including MSN file storage server (msnfs), shared home folder (usr), financial transaction processing (fin), database management system (sql) [1, 11]. The important characteristics of these traces are given in Table 1.

5 Results

In order to quantify the performance of our palloc strategies, we used IOPS (as our throughput metric) and average latency. In addition, to better understand the relationship between palloc performances and internal resource usages, we also measured the contribution of channel, way, die, and plane level parallelism to data accesses and the total number of transactions for all palloc schemes. Finally, we studied the utilization of channels and the fraction of the time spent on different internal resource activities.

5.1 Finding Overall Optimal Palloc scheme

Figures 4 and 5 plot, respectively, IOPS and average latency values for each palloc scheme tested. To enable better comparisons, all IOPS and latency numbers are *normalized* with respect to the corresponding CWDP value, which is reported as being the "optimal palloc scheme" by prior research [6, 8]. We observe that CWDP, DPWC, PWCD, and WDCP exhibit the best latency and throughput number among all channel-first, die-first, plane-first, and way-first palloc strategies, respectively. When all the test cases are considered, one can conclude that **PWCD** is the globally optimal palloc strategy from the performance angle.

From a throughput perspective, most die and plane-first palloc strategies provide about 29% better IOPS, compared to channel-first palloc schemes. One of the main reasons behind the better throughput of such strategies is that they exhibit high levels of die and plane locality, helping to build flash-transactions exploiting flash-level parallelism at on-line. Figure 6 pictorially shows the total number of flash-transactions measured at the flash chip level. Plane-first and die-first palloc strategies dramatically reduce the number of flash-transactions compared to the palloc strategies that target system-level parallelism. This is mainly because the flash-level parallelism is achieved via advance flash operations, constructed by aggregating multiple incoming requests at runtime. We observe from our experimental results that PWDC and DPWC are able to achieve 82.7% and 81.6% more flash-level parallelism, respectively, than CWDP.

The flash-level resource-first palloc schemes may introduce more bus contention in a channel when the lengths of I/O requests are not enough to span all the elements. Therefore, their latency can be worse than that of the channel-first palloc schemes. As shown in Figure 5, most die-first and plane-first palloc schemes provide 11.1% worse latency (as compared to CWDP), which are reasonable considering the significant throughput improvements they bring. Interestingly, PDCW and PDWC show even slightly lower latency compared to CWDP. This is because channel striping in some cases suffers from resource conflicts, between the committed flash operations and the current flash operations. Figure 7 presents the waiting times taken to resolve the resource conflicts. As shown in this graph, latencies for palloc schemes are as higher as the waiting time due to longest flash-transactions time. In contrast, high die-interleaving-with-multiplane operation rates (Figure 8) of PDCW and PDWC (12.9% $\sim$ 21.4%) result in reduced the overall waiting times.

A comparison of writes vs. reads. Consider a write-intensive workload (msnfs) and a read-intensive workload (fin2). For the write-intensive workload, the channel-first pallocs outperform flash-level resource-first pallocs by 23% (on average), in terms of latency, while their throughputs are on average 34.4% worse than that of the flash-level resource-first pallocs. In contrast, for the read-intensive workload, both the latency and IOPS of the channel-first pallocs are on average 12.5% and 27.9% worse than that of the flash-level resource-first pallocs, respectively. Although not presented here in detial, we

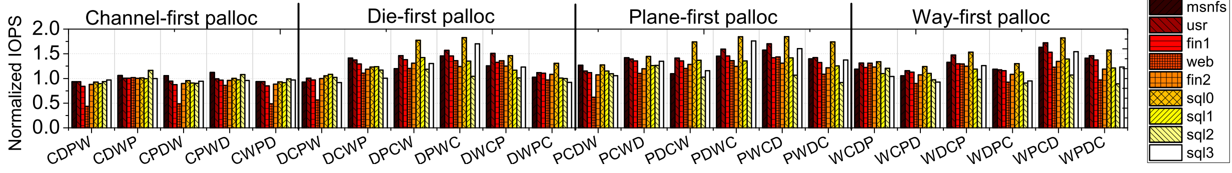


Figure 4: Throughput comparison. IOPS numbers are normalized with respect to corresponding CDPW IOPS.

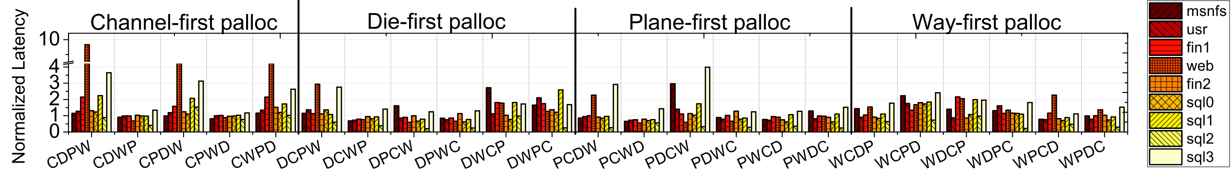


Figure 5: Latency comparison. Latency values are normalized with respect to corresponding CDPW values.

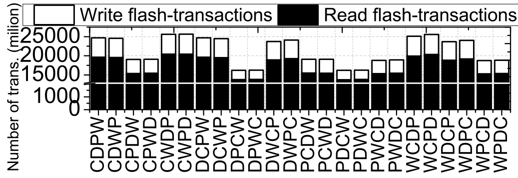


Figure 6: The number of flash-transactions executed.

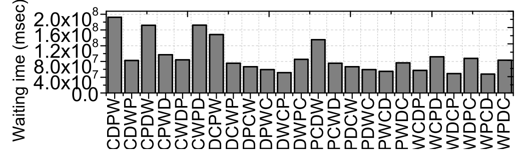


Figure 7: Waiting time required to resolve resource conflicts.

believe that one of the reasons why the channel-first pallocs show worse performance than that of a flash-level resource-first palloc in most read cases is that the bus activity fraction of the total execution time for reads, which causes high system-level resource contention, accounts for at least 50.5%, whereas that for writes is as much as 7%.

5.2 Parallelism Interference

In order to better understand the cross-interactions among parallelisms at different levels and performance, we categorize all flash-transactions executed based on their operation types. As opposed to the common perception that the system-level and flash-level concurrency mechanisms are largely orthogonal, we observe that channel striping method in system-level makes it hard to exploit flash-level parallelism. Specifically, as shown in Figure 8, the percentage of flash-level parallelism exploited by the channel-first palloc schemes shrink as much as 99.8% and on average 44.9%, compared to the plane-first palloc schemes. Even when the way-first palloc schemes are employed, the percentage of flash-level parallelism still shrinks 40.7%. The main reason behind this low flash-level parallelism is that channel and way-first palloc strategies induce poor flash-level locality. With these palloc strategies, the transfer sizes of I/Os are insufficient to span all of dies and planes, and consequently, flash-level parallel data accesses cannot be made at runtime.

5.3 Resource Utilization

Increasing resource utilization is another big concern for parallel data accesses. We observe that many internal

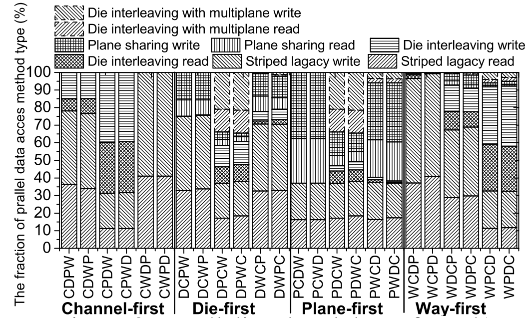


Figure 8: Parallelism breakdown for sql1.

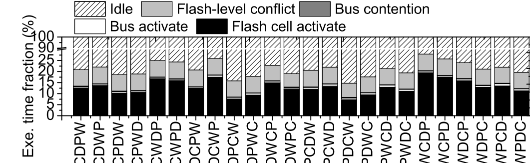


Figure 10: Execution breakdown (msnfs, write-intensive).

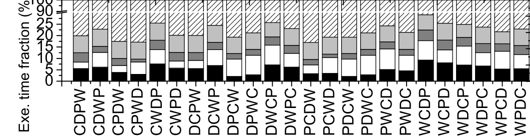


Figure 11: Execution breakdown (web, read-intensive).

resources are significantly *underutilized*. Figure 9 plots the average channel utilizations under each palloc scheme tested. The channel resource utilization accounts for 43.1% on average with most parallel data access methods. Especially, channel-first palloc schemes exhibit poor channel utilization under disk-friendly workloads even though such schemes are oriented toward taking advantage of channel-level parallelism. Since they cannot commit flash-transactions until the previous requests are completed, when there is a conflict in a flash or channel, these schemes would not be able to achieve high levels of channel utilization.

Figures 10 and 11 plot the execution time breakdown for the write and read intensive workloads, respectively. One can observe from these results that about 80% of the total execution time are spent idle.

5.4 Optimization Potential for Parallelism

Based on the I/O access patterns we studied, it can be observed that each palloc scheme exhibits different perfor-

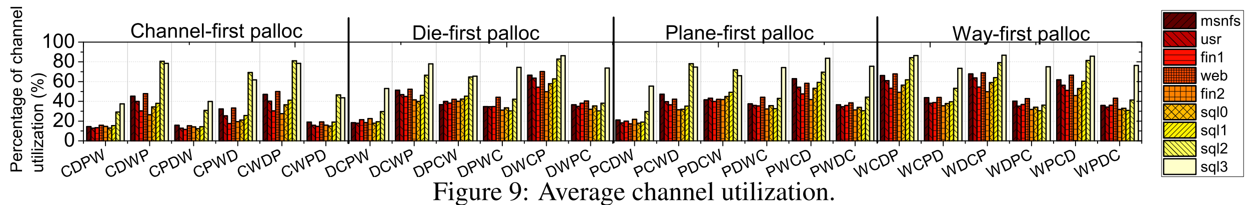


Figure 9: Average channel utilization.

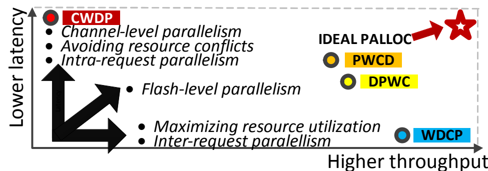


Figure 12: Performance map with optimizations points.

mance characteristics and optimization points. Figure 12 pictorially summarizes the potential of parallelism optimization from both the latency and throughput perspectives. For the latency sensitive applications, channel striping and channel-first palloc give much better position to leverage architectural parallelism. Alleviating resource conflicts is a key to reduce latency and improve inter-request parallelism. In comparison, way- and flash-level parallelism are more suitable for throughput sensitive applications. Maximizing resource utilization is a major factor in exploiting these different levels of parallelisms.

5.5 Discussion and Future Work

As our experimental results demonstrate, although flash-level resource-first palloc schemes generally perform better, their relative performances can vary based on how system-level resources are combined with flash-level resources and how well access patterns are suited to their combination. For example, PDWC favors flash-level resources more than PWCD, but the average throughput of PDWC is slightly worse than that of PWCD. We believe that this is because long data movement time of die-interleaving-with-multiplane of PDWC makes system-level resource contention a little bit more pronounced under certain workloads like usr. Similarly, the performance of WPCD is as good as DPWC. Although WPCD favors the a system-level resource (way), it allocates plane resources first among different ways within a channel, thereby achieving high flash-level parallelism with the plane sharing. Note that, in most cases that we tested, the way-first pallocs are better than the channel-first pallocs in terms of throughput. As mentioned in Section 2, the way-first pallocs reap the benefits of inter-request parallelism, which has an impact on improving bandwidth. One of the reasons behind this behavior is that flash-transactions of each I/O request are served within a channel so that several requests in the device queue can be issued over multiple channels in parallel. We however believe that the performance of the way-first pallocs would be degraded when the sizes for each request are larger than the total amount of contiguous physical pages in the channel.

We observe that, when access patterns are fully sequential, the I/O requests span all internal resources, so there is no performance difference between different pallocs.

Our on-going plans include incorporating a high-speed NAND flash interface (e.g., 400MHz). We plan to further evaluate palloc schemes with varying parameters which may have an impact on palloc performances, such as different queue/buffer managements, higher level of flash firmware, and more diverse workloads as well as micro-benchmarks.

6 Conclusion

This paper evaluates all possible page allocation (palloc) strategies using a cycle-accurate SSD simulator. Our experimental results reveal that the channel-first palloc strategies are not the best from a performance perspective, when all levels of parallelism are considered. Further, our results show that flash-level parallelism can be interfered by channel-first palloc schemes, and internal resources are significantly underutilized with most data access methods. We believe our results and observations can be used for selecting the ideal palloc schemes, given a target workload.

7 Acknowledgements

We thank our shepherd, Umesh Maheshwari, for his help and careful revisions in improving our paper. We also thank anonymous reviewers for their constructive feedback. This work is supported in part by NSF grants 1017882, 0937949, and 0833126 and DOE grant DE-SC0002156.

References

[1] http://traces.cs.umass.edu/index.php/storage.

[2] AGRAWAL, N., ET AL. Design tradeoffs for SSD performance. In *USENIX ATC* (2008).

[3] CAULFIELD, A. M., ET AL. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ASPLOS* (2009).

[4] CHEN, F., ET AL. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *HPCA* (2011).

[5] DIRIK, C., AND JACOB, B. The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization. In *ISCA* (2009).

[6] HU, Y., ET AL. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *ISC* (2011).

[7] INTEL, AND SEAGATE. Serial ATA NCQ.

[8] JUNG, J.-Y. S., ET AL. FTL design exploration in reconfigurable high-performance SSD for server applications. In *ICS* (2009).

[9] JUNG, M., ET AL. NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level. In *MSST* (2012).

[10] MICRON, INC. MT29F8G08MAAWC.

[11] NARAYANAN, D., ET AL. Migrating server storage to SSDs: Analysis of tradeoffs. In *EuroSys* (2009).

[12] PARK, S.-H., ET AL. Design and analysis of flash translation layers for multi-channel NAND flash-based storage devices. In *TCE* (2009).

[13] YOO, B., ET AL. SSD characterization: From energy consumption's perspective. In *HotStorage* (2011).